ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Χρονοπρογραμματισμός διανομής για ένα και συνεργαζόμενα Μη Επανδρωμένα Ιπτάμενα Οχήματα

Delivery scheduling for single and cooperative Unmanned Aerial Vehicles

Διπλωματική εργασία

Στρατή Ιωάννα

Επιβλέποντες: Κατσαρός Δημήτριος
Επίκουρος Καθηγητής Π.Θ
Κοράκης Αθανάσιος
Επίκουρος Καθηγητής Π.Θ

Βόλος 2018

Στην οικογένεια μου και τους φίλους μου.

Ευχαριστίες

Με την περάτωση της παρούσας εργασίας θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Κατσαρό Δημήτριο για την εμπιστοσύνη που έδειξε, για την άριστη συνεργασία καθώς και για τη συνεχή καθοδήγηση που διευκόλυναν την εκπόνηση της Διπλωματικής μου εργασίας.
Επίσης θα ήθελα να ευχαριστήσω τον συνεπιβλέποντα καθηγητή κ. Κοράκη Αθανάσιο.
Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου για την αμέριστη υποστήριξη και τη βοήθεια που μου παρείχαν καθ' όλη τη διάρκεια των σπουδών μου, καθώς και στους φίλους μου για τις αξέχαστες στιγμές.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη αλγορίθμων για τον προγραμματισμό μη επανδρωμένων εναέριων οχημάτων (drone) ώστε να μεταφέρουν πακέτα σε προορισμούς. Πιο συγκεκριμένα, αρχικά γίνεται προσομοίωση του οδικού δικτύου της περιοχής και δημιουργείται ένα γράφημα, όπου εκεί μπορεί να κινείται το φορτηγό που μεταφέρει τα πακέτα και το drone. Το φορτηγό σταθμεύει σε κάποιον κόμβο που είναι κοντά στον προορισμό του πακέτου και από εκεί ξεκινά το ταξίδι του drone για την παράδοση του πακέτου. Όταν παραδοθούν όλα τα πακέτα έχει συγκεντρωθεί πληροφορία για την ενέργεια που κατανάλωσε το drone, πόσες ώρες πτήσεις συμπλήρωσε και πόσα πακέτα παραδόθηκαν σε μία ώρα. Αυτά τα στοιχεία θα παρασταθούν γραφικά ώστε να μελετηθεί η επίδοση των αλγορίθμων.

Παρουσιάζεται αναλυτικά η διαδικασία που ακολουθήθηκε για την υλοποίηση των αλγορίθμων καθώς και ψευδοκώδικας για κάθε αλγόριθμο. Το πρόγραμμα γράφηκε κυρίως σε γλώσσα C χρησιμοποιώντας κάποιες δομές από τη C++.

Abstract

The purpose of the present diploma thesis is the algorithm development for the scheduling Unmanned Aerial Vehicles (drone) in order to deliver packets in destinations. More precisely, at first the road network of the area is simulated and a graph is created, where the drone carrier (which transports the packets and the drone) is moving. The drone carrier parks at a node which is close to the packet destination and from there the drone begins the journey for the packet delivery. When all the packets have been delivered, information has been gathered for drone energy consumption, drone flight time and packets throughput. These elements will be displayed graphically in order to study the performance of the algorithms.

It is presented in detail the process followed for the algorithm development and the pseudocode for each algorithm. The program was written mainly in programming language C with some structures from C++.

Περιεχόμενα

1. Εισαγωγή

1.1 Μη Επανδρωμένα Ιπτάμενα Οχήματα (UAVs)

Ένα μη επανδρωμένο ιπτάμενο όχημα (Unmanned Aerial Vehicle (UAV)) κοινώς γνωστό σαν drone είναι ένα αεροσκάφος χωρίς χειριστή στην άτρακτό του. Τα UAVs είναι μια συνιστώσα του συστήματος μη επανδρωμένου αεροσκάφους (UAS), το οποίο περιλαμβάνει ένα UAV, έναν ελεγκτή στο έδαφος και ένα σύστημα επικοινωνίας μεταξύ τους. Η πτήση UAVs μπορεί να γίνει σε διαφορετικούς βαθμούς αυτονομίας, είτε με απομακρυσμένο έλεγχο από ανθρώπινο χειριστή ή αυτόνομα με ενσωματωμένους υπολογιστές. [1]

Τα είδη των UAVs χωρίζονται σε δύο βασικές κατηγορίες:

- Με σταθερές πτέρυγες όπως τα αεροπλάνα
- Με έλικες όπως τα ελικόπτερα

Για τον προσδιορισμό του αριθμού των κινητήρων ή των ελίκων χρησιμοποιείται ο όρος Multicopter ή Multirotor και διαχωρίζονται σε Quadcopter (με 4 έλικες), Hexacopter, Octacopter κλπ.

Ένα quadcopter ανυψώνεται και προωθείται από τέσσερις ρότορες. Τα quadcopters χρησιμοποιούν δύο ζευγάρια πανομοιότυπων ελικοειδών κινητήρων, δύο δεξιόστροφους και δύο αριστερόστροφους. Χρησιμοποιούν την ανεξάρτητη μεταβλητή της ταχύτητας κάθε ρότορα για να πετύχουν έλεγχο. Με την αλλαγή της ταχύτητας κάθε ρότορα είναι δυνατόν να παραχθεί μια συνολική ώθηση, για τον εντοπισμό του κέντρου ώθησης και πλευρικά και κατά μήκος και για να δημιουργηθεί η επιθυμητή συνολική ροπή ή η δύναμη περιστροφής. [2]

Στα τέλη της δεκαετίας του 2000s οι πρόοδοι στην ηλεκτρονική επέτρεψαν την παραγωγή φθηνών, ελαφριών ελεγκτών πτήσεων όπως accelerometers, global positioning system (GPS) και κάμερες. Ένα accelerometer είναι μια συσκευή η οποία μετράει την κατάλληλη επιτάχυνση. Κατάλληλη επιτάχυνση είναι η επιτάχυνση ενός σώματος σε σημείο στιγμιαίας ηρεμίας.
Αυτά τα quadcopters λόγω του μικρού μεγέθους τους και της ικανότητας ελιγμών, είναι κατάλληλα για πτήσεις τόσο σε εσωτερικούς όσο και σε εξωτερικούς χώρους.

Για την παροχή ενέργειας για τα μικρά UAVs συνήθως χρησιμοποιούνται μπαταρίες Lithium-Polymer (Li-Po), οι οποίες είναι επαναφορτιζόμενες

τεχνολογίας lithium-ion. Τα μεγαλύτερα οχήματα βασίζονται σε συμβατικές μηχανές αεροπλάνων. Η κλίμακα ή το μέγεθος του αεροσκάφους δεν αποτελεί το καθοριστικό χαρακτηριστικό της παροχής ενέργειας για ένα UAV. Προς το παρόν, η πυκνότητα ενέργειας των μπαταριών Li-Po είναι πολύ μικρότερη από της βενζίνης. Το ρεκόρ ταξιδιού ενός UAV σε ολόκληρο τον βόρειο Ατλαντικό ωκεανό κατέχει ένα αεροσκάφος τύπου βενζίνης. Το ρεκόρ αυτό έχει το UAV που πέταξε για 1882 μίλια με λιγότερο από ένα γαλόνι βενζίνης, δηλαδή με μόλις 3.78 λίτρα. Η ηλεκτρική ενέργεια χρησιμοποιείται γιατί απαιτείται λιγότερη δουλειά για την πτήση και οι ηλεκτρικοί κινητήρες είναι πιο ήσυχοι. Επίσης, κατάλληλος σχεδιασμός της αναλογίας ώθησης προς βάρος σε έναν ηλεκτρικό κινητήρα ή κινητήρα βενζίνης που κινεί μια έλικα μπορεί να κάνει το UAV να είναι σταθερό σε ένα σημείο στον αέρα ή να ανέβει κατακόρυφα.

Εικόνα 1: Quadcopter

Εικόνα 2: Hexacopter

Εικόνα 3: Octacopter

1.2 Εφαρμογές των UAVs

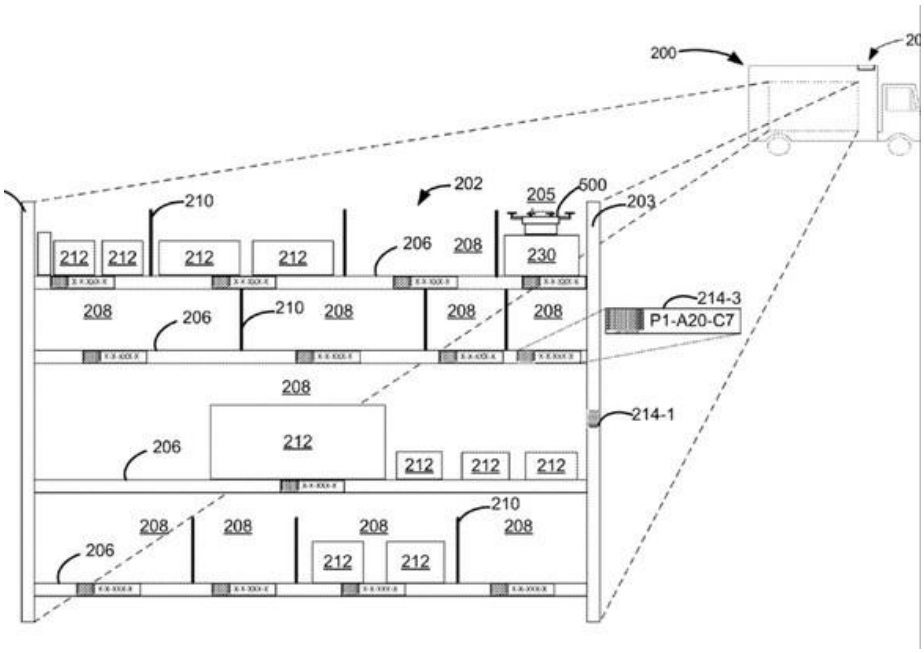
Τα UAVs χρησιμοποιούνται σε διάφορους τομείς:

- Αεροδιαστημική: Τα UAVs χρησιμοποιούνται για την συντήρηση και την επισκευή αεροσκαφών. Τον Ιούνιο του 2015 η EasyJet ξεκίνησε να τεστάρει τα UAVs για την συντήρηση των Airbus A320s και τον Ιούλιο του 2016 η Airbus κατέδειξε την χρήση των UAVs για την οπτική επιθεώρηση αεροσκαφών. Ωστόσο, κάποιοι ειδική στην συντήρηση αεροσκαφών παραμένουν επιφυλακτικοί όσον αφορά την τεχνολογία και την ικανότητα της να συλλαμβάνει πιθανούς κινδύνους.
- Στρατός: Τα UAVs χρησιμοποιούνται από ένα ευρύ φάσμα στρατιωτικών δυνάμεων για αναγνώριση, επίθεση, άμυνα έναντι άλλων UAVs και στρατιωτική εκπαίδευση.
- Δημόσια: Η δημόσια χρήση περιλαμβάνει εναέριες έρευνες για καλλιέργειες, εναέριες φωτογραφίες, επιθεώρηση των γραμμών ηλεκτρικής ενέργειας και των αγωγών, μελέτη της άγριας φύσης, μεταφορά φαρμάκων σε μη προσβάσιμες περιοχές, ανίχνευση παράνομου κυνηγιού ζώων, λειτουργίες αναγνώρισης, παρακολούθηση περιβάλλοντος, ανίχνευση πυρκαγιάς και παρακολούθηση, συντονισμό ανθρωπιστικής βοήθειας, μέτρηση κατολίσθησης, παράνομη ανίχνευση αποβλήτων, λαθρεμπόριο και παρακολούθηση πλήθους.
- Χόμπι και ψυχαγωγική χρήση: Τα UAVs χρησιμοποιούνται για φωτογράφηση και βιντεοσκόπηση.
- Δημοσιογραφία: Χρησιμοποιούνται για την συλλογή ειδήσεων.
- Έρευνα και διάσωση: Χρησιμοποιούνται για την αναζήτηση αγνοούμενων ανθρώπων, σαν ναυαγοσώστες εντοπίζοντας κολυμβητές που βρίσκονται σε κίνδυνο χρησιμοποιώντας θερμικές κάμερες και ρίχνοντας τους σωσίβια.
- Επιστημονική έρευνα: Χρησιμοποιούνται για την έρευνα επικίνδυνων φαινομένων όπως για παράδειγμα ένας τυφώνας.

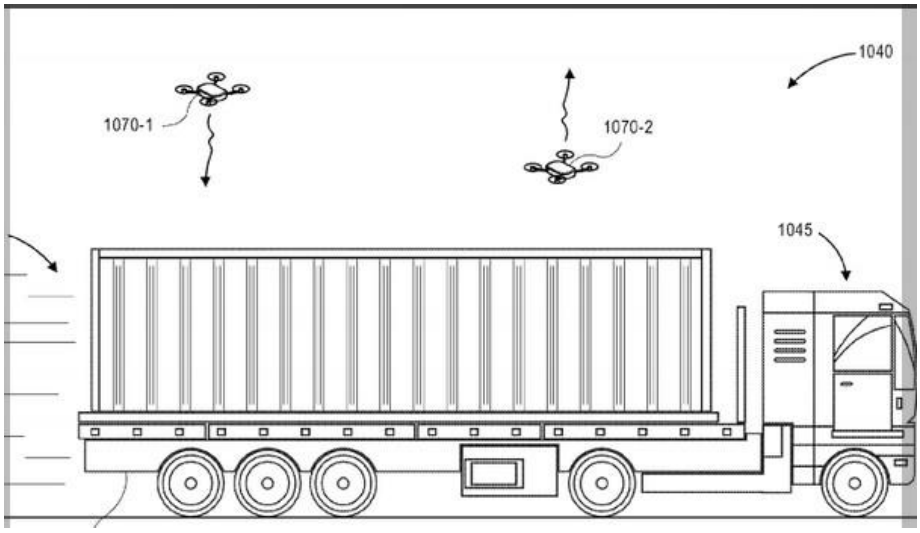
1.3 Κίνητρο για την μελέτη των UAVs

Η Amazon έχει δημιουργήσει μια αυτόνομη τεχνολογία φορτηγών που μπορεί να λειτουργήσει με UAVs για την παράδοση πακέτων γρήγορα και αποτελεσματικά. Αυτή η τεχνολογία αφορά φορτηγά χωρίς οδηγό (που μπορεί επίσης να είναι αυτοκίνητα, αεροπλάνα, ή πλοία), ονομάζονται κινητές βάσεις και περιφέρονται γύρω από μια πόλη με πακέτα προς παράδοση. Αντί να οδηγούν σε κάθε διεύθυνση για την παράδοση των πακέτων, τα οχήματα χωρίς οδηγό μπορούν να είναι εξοπλισμένα με αυτοματοποιημένα συστήματα αποθήκευσης και ανάκτησης ώστε τα πακέτα να φορτωθούν σε drone για το τελευταίο σκέλος της παράδοσης. Με αυτό τον τρόπο το κόστος παράδοσης μειώνεται και τα πακέτα φτάνουν στον παραλήπτη γρηγορότερα. [3]

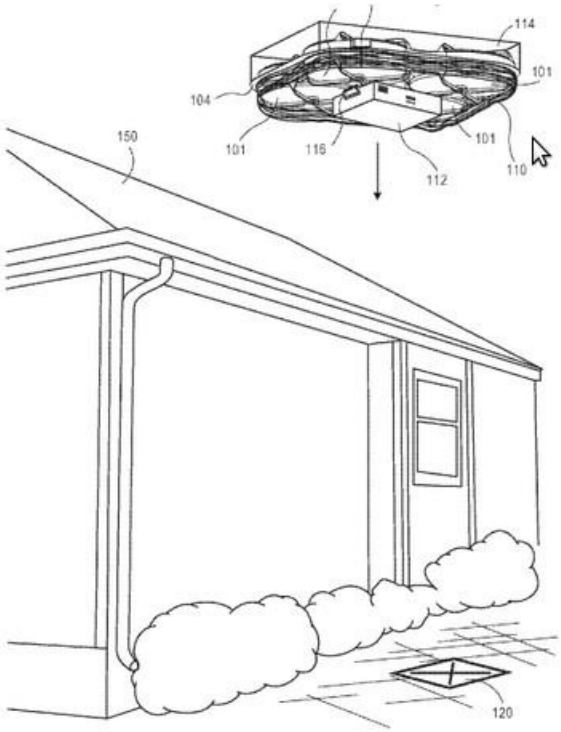
Η χρήση drone που κινούνται με ηλεκτρική ενέργεια για τη διανομή πακέτων είναι καλύτερη για το περιβάλλον γιατί μειώνοντας την ανάγκη της διανομής με φορτηγά εξοικονομούνται καύσιμα και μειώνονται οι εκπομπές άνθρακα στην ατμόσφαιρα. [4]
Το ποσό ενέργειας που καταναλώνει ένα drone εξαρτάται από το πόσο βαρύ είναι το ίδιο, από τα πακέτα που μεταφέρει, από τις μπαταρίες του αλλά από άλλους παράγοντες όπως η ταχύτητά του και οι καιρικές συνθήκες.



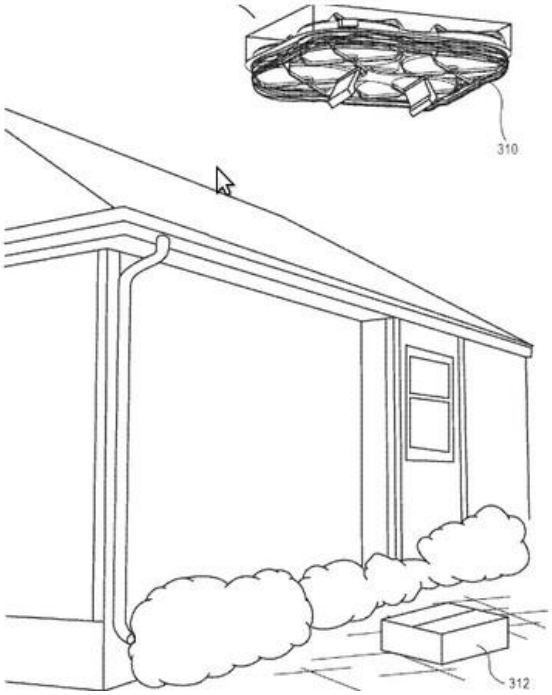
Εικόνα 4: Το φορτηγό χωρίς οδηγό με τα πακέτα προς παράδοση.



Εικόνα 5: Το drone με το πακέτο ξεκινάει προς τον προορισμό.



Εικόνα 6: Το drone φτάνει στον προορισμό του για την παράδοση του πακέτου.



Εικόνα 7: Παράδοση του πακέτου.

2. Περιγραφή προβλήματος

2.1 Διανομή πακέτων με UAVs - Γενική ιδέα

Για την διανομή πακέτων σε διάφορους προορισμούς θα γίνει ανάπτυξη ενός συστήματος που ένα φορτηγό και ένα UAV συνεργάζονται. Το φορτηγό θα μπορεί να κινείται στο οδικό δίκτυο και να πλησιάζει τις περιοχές όπου υπάρχουν πακέτα προς παράδοση. Το φορτηγό θα σταθμεύει σε κάποιο σημείο και εκεί θα γίνεται η ανάθεση στο UAV να μεταφέρει το πακέτο μέχρι τον τελικό προορισμό. Το UAV θα μπορεί να μεταφέρει ένα πακέτο τη φόρα και μόλις ολοκληρώσει αυτή την ενέργεια θα επιστρέφει στο φορτηγό για να προετοιμαστεί για την επόμενη διαδρομή. Η ενέργειά του θα προέρχεται από μπαταρίες οι οποίες είναι επαναφορτιζόμενες. Επίσης, θα υπάρχουν πάντα επιπλέον μπαταρίες σε περίπτωση που η ενέργεια του UAV τελειώσει ώστε να αντικατασταθούν άμεσα και να μην υπάρξει σημαντική καθυστέρηση στην διανομή των πακέτων.

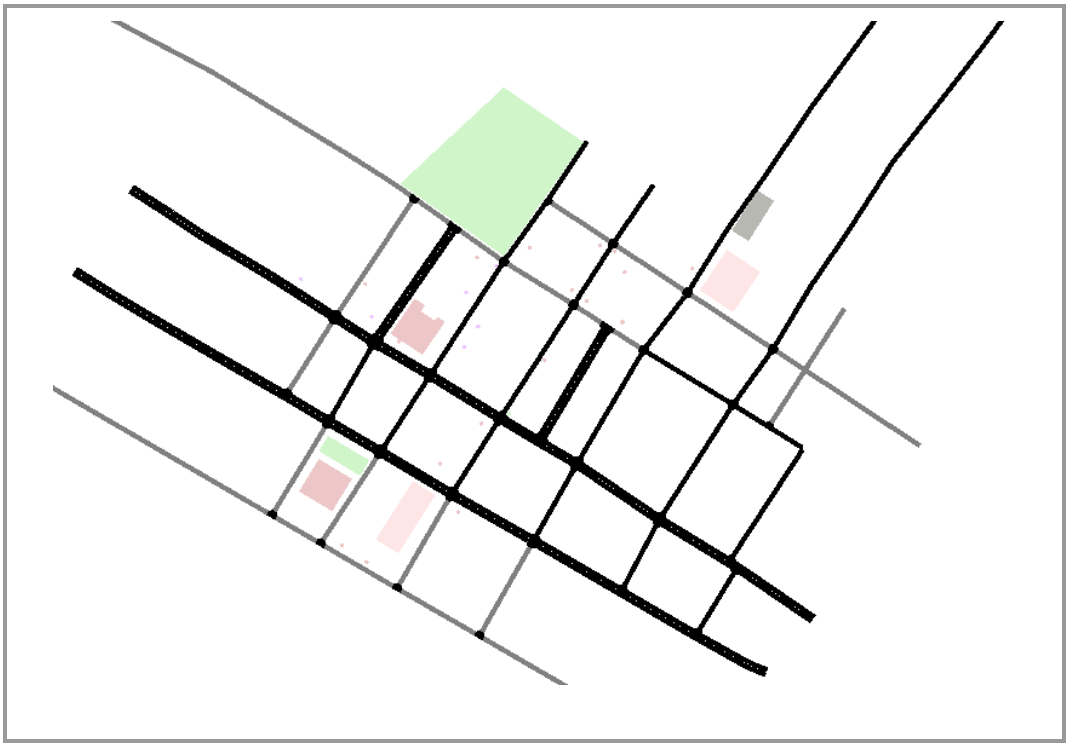
3. Περιβάλλον προσομοίωσης

3.1 Προσομοίωση οδικού δικτύου - SUMO simulator

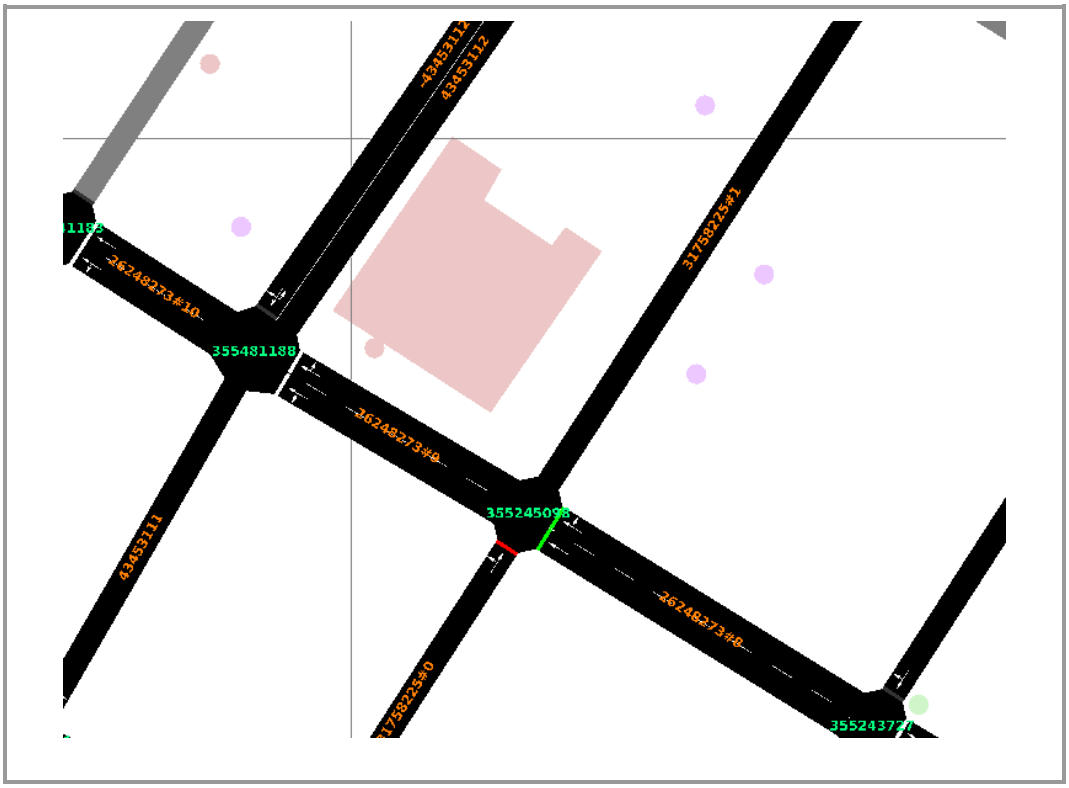
Η προσομοίωση του οδικού δικτύου έγινε με το "SUMO simulator". Το SUMO (Simulation of Urban MObility) είναι ένα δωρεάν και ανοιχτό λογισμικό κατάλληλο για την προσομοίωση της οδικής κυκλοφορίας, σχεδιασμένο για να χειρίζεται μεγάλα οδικά δίκτυα.[5] Αρχικά, έγινε εγκατάσταση του SUMO σε Linux. Επίσης, για τη λειτουργία του SUMO απαιτείται εγκατάσταση της Python. Στη συνέχεια, χρειάστηκαν δεδομένα για το χάρτη της περιοχής που θα γίνει προσομοίωση από το OpenStreetMap.[7] Αφού προσδιοριστούν τα όρια της περιοχής γίνεται download ενός αρχείου "map.osm". Έπειτα, γίνεται μετατροπή του αρχείο από "map.osm" σε "map.net.xml" όπου εκεί υπάρχει όλη η πληροφορία που χρειάζεται για τις διασταυρώσεις, τους δρόμους και τη μεταξύ τους διασύνδεση. Το λογισμικό διαθέτει gui (graphical user interface) που αναπαριστά όλη την πληροφορία που υπάρχει στο xml αρχείο με τα ίδια ονόματα στους δρόμους και τις διασταυρώσεις, όπως φαίνεται στις εικόνες 1 και 2 παρακάτω. Η διαδικασία και οι εντολές που χρησιμοποιήθηκαν για την προσομοίωση βρίσκονται αναλυτικά στο wiki του SUMO.[6] Ενδεικτικά το αρχείο έχει την μορφή:

```
<edge id="-406851250" from="4089071819" to="4089071818" priority="2"
type="highway.service">
        <lane id="-406851250_0" index="0" allow="delivery" speed="5.56" length="76.80"
        shape="627.99,476.33 588.85,410.24"/>
</edge>
...
<tlLogic id="355243713" type="static" programID="0" offset="0">
      <phase duration="31" state="GG"/>
      <phase duration="9" state="yy"/>
</tlLogic>
...
<junction id="354799645" type="unregulated" x="329.37" y="565.81"
incLanes="26248273#12_0 26248273#12_1" intLanes="" shape="331.12,568.54
327.61,563.07"/>
...
<connection from="40786463#2" to="40786463#3" fromLane="1" toLane="1"
via=":355245097_1_1" dir="s" state="M"/>
```

Εικόνα 8: Μορφή xml αρχείου.



Εικόνα 9: Παράδειγμα χρήσης του SUMO για την προσομοίωση μιας μικρής περιοχής της πόλης του Βόλου.



Εικόνα 10: Αναγραφή ονομάτων των δρόμων και των junctions όπως αναφέρονται στο xml αρχείο.

3.2 Δημιουργία γραφήματος

Για τη δημιουργία του γραφήματος χρησιμοποιήθηκε η πληροφορία του xml αρχείου. Η διαδικασία που ακολουθήθηκε ήταν parsing του xml αρχείου ώστε να απομονωθούν τα στοιχεία που χρειάζονται. Πρώτα έγιναν parse τα edges (δρόμοι) όπου τα στοιχεία που κρατήθηκαν ήταν το "id", το "from" και το "to". Το "from" δηλώνει από ποιο junction ξεκινάει το κάθε edge και το "to" σε ποιο junction καταλήγει. Στη συνέχεια έγιναν parse τα lanes (λωρίδες) όπου κρατήθηκαν το "id" και το "length". Το "id" κάθε lane υποδηλώνει σε ποιο edge ανήκει και πόσα lanes έχει κάθε edge καθώς η μορφή του είναι "-406851250_0", το "-406851250" είναι το edge id και _0 δηλώνει ότι είναι το lane 0. Τέλος, έγιναν parse τα junctions (διασταυρώσεις / κόμβοι) όπου κρατήθηκαν το "id", το "x" και το "y". Τα "x" και "y" είναι οι συντεταγμένες κάθε junction στο δισδιάστατο χώρο.

Η διαδικασία για το parsing των edges, των lanes και των junctions έγινε με τρεις συναρτήσεις με ονόματα edges_separator(), lanes_separator() και junctions_separator() αντίστοιχα. Ενδεικτικά, ένα παράδειγμα ψευδοκώδικα για το πως έγινε ο διαχωρισμός των στοιχείων από το xml αρχείο:

```
edge_separator() {

   s =  read_one_line(); //reads one line from xml file with tag = edge
   pos1 = find_substring(id=");
   pos2 = find_substring(");
   edge_id =substring( pos2-pos1);

}
```

Εικόνα 11: Ψευδοκώδικας για τον διαχωρισμό των στοιχείων από το xml αρχείο.

Έπειτα έγινε κλήση αυτών των συναρτήσεων και τα αποτελέσματα της edges_separator() αποθηκεύτηκαν σε ένα διάνυσμα ενώ τα αποτελέσματα της lanes_separator() και της junctions_separator() σε δομές map. Σε μια δομή map αποθηκεύονται στοιχεία που σχηματίζονται από ένα συνδυασμό κλειδιού και μιας τιμής map ακολουθώντας μια συγκεκριμένη σειρά. Στο map οι τιμές των κλειδιών χρησιμοποιούνται για να ταξινομήσουν και να προσδιορίσουν με μοναδικό τρόπο τα στοιχεία, ενώ οι τιμές map αποθηκεύουν το περιεχόμενο που σχετίζεται με το κλειδί. Στην περίπτωση αυτή η δομή map χρησιμοποιήθηκε γιατί εξυπηρετεί στο να γίνεται καλύτερα η αναζήτηση, λόγω ότι η πληροφορία είναι ταξινομημένη. Στη συνέχεια, αφού έχει συγκεντρωθεί όλη η πληροφορία που απαιτείται, σειρά έχει η σύνδεση όλων αυτών των στοιχείων. Για κάθε edge που υπάρχει γίνεται αναζήτηση του from και του to μέσα στο map με τα junctions. Εάν υπάρχουν για κάθε lane γίνεται αναζήτηση της θέσης του μέσα στο map με τα lanes και σε αυτή τη θέση αποθηκεύονται σε ένα διάνυσμα τα lanes που εξέρχονται από κάθε junction. Έπειτα δημιουργείται και ένα διάνυσμα που περιέχει τα lanes που εισέρχονται σε κάθε junction. Έτσι γνωρίζουμε για κάθε κόμβο ποιοι δρόμοι εισέρχονται και ποιοι εξέρχονται από αυτόν και αυτό είναι ουσιαστικά που χρειάζεται για τη διασύνδεση του γραφήματος. Παρακάτω παρατίθεται ο ψευδοκώδικας αυτής της διαδικασίας:

```
for each edge {
   from_it = search_junctionMap_for_from_junction();
   to_it = search_junctionMap_for_to_junction();

   if (from_it exists && to_it exists){
      for each lane of the edge{
         lane_it = search_laneMap_for_edge->lane();

         //create outLanes
         if (lane_it exists){
            lane_it_from = from_it;
            lane_it_to = to_it;
            outLanes[] = lane_it;

            //create backLanes
            lane new_lane;
            new_lane_id = lane_it_id;
            new_lane_length = lane_it_length;
            new_lane_from = to_it;
            new_lane_to = from_it;
            backLanes[] = new_lane;
         }
      }
   }
}
```

Εικόνα 12: Ψευδοκώδικας δημιουργίας του γραφήματος.

3.3 Δημιουργία προορισμών και πακέτων

Για τη δημιουργία προορισμών χρειάζεται να προσδιοριστούν η μέγιστη και η ελάχιστη τιμή των x και y. Αυτό γίνεται διατρέχοντας τις συντεταγμένες των junctions και αναζητώντας τις ελάχιστες τιμές τους. Σε κάθε προορισμό αντιστοιχεί ένα πακέτο προς παράδοση που οι τιμές του κυμαίνονται από 0.1 kg έως 25 kg.
Η κατανομή προορισμών και πακέτων στο χώρο έγινε με δύο τρόπους:

- uniform με τη χρήση της srand().
- πολωμένη με τη χρήση της Zipfian κατανομής.

Για την δημιουργία τυχαίων ομοιόμορφα κατανεμημένων προορισμών και των αντίστοιχων βαρών χρησιμοποιήθηκε η παρακάτω συνάρτηση:

```
double rand_min_to_max(double Min, double Max){

    return Min + (rand() / ( RAND_MAX / (Max-Min) ) ) ;
}
```

Εικόνα 13: Συνάρτηση rand_min_to_max, παράγει τυχαίους αριθμούς μεταξύ min και max.

Η συνάρτηση αυτή καλείται δύο φορές, μία για την παραγωγή τυχαίου αριθμού x και μία για την παραγωγή τυχαίου αριθμού y, μέσα σε μία επανάληψη όπου εκτελείται για N φορές, όπου N είναι ο αριθμός των προορισμών που θα δημιουργηθούν. Στη συνέχεια για κάθε προορισμό πρέπει να δημιουργηθεί ένα πακέτο όπου πάλι καλείται η παραπάνω συνάρτηση με Min το ελάχιστο βάρος του πακέτου και Max το μέγιστο βάρος του.

Για την παραγωγή πολωμένων προορισμών με χρήση της Zipfian κατανομής αρχικά πρέπει να χωριστούν οι περιοχές από Xmin μέχρι Xmax και Ymin μέχρι Ymax σε P ίσα μέρη. Στη συνέχεια γίνεται κλήση της συνάρτησης zipf(), η οποία παράγει αριθμούς μεταξύ 1 και P, δύο φορές μία για το διάστημα [Xmin, Xmax] και μια για το [Ymin, Ymax], μέσα σε μία επανάληψη που εκτελείται N φορές, όπου N ο αριθμός των προορισμών που θα δημιουργηθούν. Στη συνάρτηση zipf() η παράμετρος θ παίρνει τιμές στο διάστημα [0...1]. Έπειτα, για κάθε προορισμό γίνεται παραγωγή ενός πακέτου με την ίδια διαδικασία, δηλαδή χωρίζοντας σε P ίσα μέρη το διάστημα από το ελάχιστο βάρος μέχρι το μέγιστο, και καλώντας την συνάρτηση zipf(). Παρακάτω, υπάρχει ο ψευδοκώδικας παραγωγής zipfian τιμής:

```
function zipf(theta, parts) {

  loop i from 1 to parts {
    c = c + (1.0 / i^theta);
  }

  c = 1.0 / c;
  sum_probs[0] = 0;

  loop i from 1 to parts{
    sum_probs[i] = sum_probs[i-1] + c / i^theta;
  }

  do{
    z = rand(0,1);
  }while (z == 0 || z == 1);

  low = 1;
  high = parts;

  do{
    mid = (low+high) / 2;
    if(sum_probs[mid] > z && sum_probs[mid-1] < z) {
      zipf_value = mid;
    }else if(sum_probs >= z){
      high = mid - 1;
    }else{
      low = mid + 1;
    }
  }while( low <= high);

  if(zipf value is not between 1 and parts){
    print ("Zipfian value out of range");
  }
   return (zipf_value);

}
```

Εικόνα 14: Ψευδοκώδικας συνάρτησης zipf, παράγει τυχαίους αριθμούς μεταξύ 1 και parts.

3.4 Εύρεση συντομότερου μονοπατιού στο οδικό δίκτυο

Έχοντας ολοκληρώσει τις διαδικασίες για τη δημιουργία του γραφήματος του οδικού δικτύου και την δημιουργία προορισμών, σειρά έχει ένας αλγόριθμος ο οποίος θα βρίσκει ποια είναι η συντομότερη διαδρομή από έναν τρέχον κόμβο προς τον κόμβο προορισμό. Η διαδικασία εύρεσης του κόμβου προορισμού θα συζητηθεί στη συνέχεια. Ο αλγόριθμος που χρησιμοποιήθηκε είναι ο αλγόριθμος Dijkstra, ο οποίος είναι εύρεσης συντομότερου μονοπατιού σε ένα γράφημα. Στην περίπτωση αυτή το γράφημα είναι κατευθυνόμενο με μη αρνητικά βάρη στις ακμές. Ο αλγόριθμος του Dijkstra είναι άπληστος, δηλαδή σε κάθε βήμα βρίσκει τοπικά βέλτιστη λύση, ώσπου στο τελευταίο βήμα συνθέτει μια συνολικά βέλτιστη λύση. Ο αλγόριθμος αυτός επιλέχθηκε με το κριτήριο ότι βρίσκει τα μονοπάτια που πρέπει να ακολουθήσουμε από ένα κόμβο αφετηρία προς τους υπόλοιπους με το λιγότερο δυνατό κόστος. Άλλοι αλγόριθμοι που θα μπορούσαν να χρησιμοποιηθούν είναι των Bellman - Ford ή των Floyd - Warshall, αλλά αυτοί είναι πιο αποδοτικοί σε γραφήματα που έχουν αρνητικά βάρη στις ακμές και σε αυτό το γράφημα δεν γίνεται να υπάρχουν αρνητικά βάρη καθώς τα βάρη υποδηλώνουν αποστάσεις.[8]

Περιγραφή του αλγορίθμου:
Δεδομένου ενός γραφήματος G(V,E), όπου V το σύνολο των κόμβων και E το σύνολο των ακμών του. Επίσης έχουμε μια συνάρτηση βάρους ορισμένη στις ακμές του γράφου, δηλαδή για τη μετάβαση από έναν κόμβο του γραφήματος σε έναν άλλο θα υπάρχει κάποιο κόστος.
Για τη λειτουργία του αλγορίθμου, σε ένα διάνυσμα d[], μεγέθους |V|=n αποθηκεύεται η έως τώρα υπολογισμένη απόσταση των κόμβων από την αφετηρία. Κατά την αρχικοποίηση οι αποστάσεις θέτονται ως d[s] = 0 και d[v] = ∞, όπου s ≠ v και το s είναι ο κόμβος πηγή. Επιπλέον, ο αλγόριθμος διατηρεί μια ουρά προτεραιότητας Q, ώστε οι κόμβοι του γραφήματος να επεξεργάζονται με τη σωστή σειρά και ένα σύνολο S όπου περιέχει τους κόμβους που ο αλγόριθμος δεν έχει βρει ελάχιστη διαδρομή. Στην Q εισάγονται όλοι οι κόμβοι του γραφήματος με κλειδί το διάνυσμα των αποστάσεων d[], ενώ το σύνολο S είναι αρχικά κενό. Τέλος, ο αλγόριθμος χρησιμοποιεί ακόμη ένα διάνυσμα το prev[] στο οποίο για κάθε κόμβο u αποθηκεύεται ο αμέσως προηγούμενος κόμβος στο ελάχιστο μονοπάτι προς τον u. Το διάνυσμα prev[] αρχικοποιείται με null.
Μετά την αρχικοποίηση των παραπάνω δομών, ο αλγόριθμος εξάγει από την ουρά τον κόμβο με την ελάχιστη απόσταση και τον εισάγει στο S. Στο

πρώτο βήμα, ο αλγόριθμος θα εξάγει τον κόμβο αφετηρία x γιατί έχει απόσταση 0 ενώ όλοι υπόλοιποι κόμβοι έχουν άπειρη απόσταση. Για κάθε γείτονα y (του x) υπολογίζεται ένα ελαφρύτερο μονοπάτι από το ήδη υπολογισμένο και καταχωρείται στο διάνυσμα d[]. Με την αλλαγή του d[y] αλλάζει και η θέση του κόμβου y στην Q. Αφού ο αλγόριθμος εξετάσει όλους τους γείτονες του x που δεν ανήκουν στο S, εισάγει στο S τον κόμβο με την μικρότερη απόσταση. Έπειτα, ο αλγόριθμος επιλέγει τον επόμενο κόμβο από την ουρά Q και επαναλαμβάνει την παραπάνω διαδικασία μέχρι να αδειάσει η ουρά.
Όταν πλέον αδειάσει η ουρά ο αλγόριθμος έχει βρει τα ελάχιστα μονοπάτια από τον κόμβο s προς όλους τους υπόλοιπους κόμβους και τα κόστη τους.

```
function Dijkstra (graph, source){
  create_queue Q;

  for each junction v in graph{
    dist[v] = infinity;
    prev[v] = null;
    add v to Q;
  }

  dist[source] = 0;

  while Q is not empty{
    u = extract_min(Q);
    for each neighbor v of u{
      alt = dist[u] + length(u,v);

      if (alt < dist[v]) {
        dist[v] = alt;
        prev[v] = u;
      }
    }
  }

  return dist[], prev[];
}
```

Εικόνα 15: Ψευδοκώδικας συνάρτησης Dijkstra.

3.5 Εύρεση κόμβου - προορισμού

Για την εύρεση του κόμβου - προορισμού, δηλαδή του προορισμού για το φορτηγό (drone carrier) χρειάστηκε η υλοποίηση μιας συνάρτησης στην οποία δεδομένου ενός προορισμού για την παράδοση πακέτου πρέπει να βρεθεί ο πλησιέστερος κόμβος σε αυτό το σημείο. Σε αυτή τη συνάρτηση χρησιμοποιήθηκε και πληροφορία από τον αλγόριθμο Dijkstra και πιο συγκεκριμένα το διάνυσμα με τις αποστάσεις d[], ώστε να γίνεται έλεγχος μόνο σε κόμβους όπου υπάρχει συντομότερο μονοπάτι προς αυτούς και δεν έχουν άπειρη απόσταση από τον κόμβο - αφετηρία. Έτσι, έχοντας γίνει έλεγχος ότι υπάρχει μονοπάτι για κάθε κόμβο που διατρέχεται γίνεται υπολογισμός της Ευκλείδειας απόστασης όλων τον κόμβων με το σημείο προορισμό του πακέτου και γίνεται αναζήτηση της ελάχιστης απόστασης. Ο μαθηματικός τύπος υπολογισμού της Ευκλείδειας απόστασης είναι ο εξής:

$$Euclidean\ distance = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Ο κόμβος που έχει την μικρότερη απόσταση από τον προορισμό του πακέτου θα αποτελέσει τον κόμβο - προορισμό του φορτηγού. Ο ψευδοκώδικας για την εύρεση του κόμβου προορισμού έχει ως εξής:

```
function find_closest_junction_from_destination() {
  loop i from 1 to d.size() {
    if(d[i] != infinity) {
        for each junction {
          calculate_euclidean_distance(destination, junction);
          find_min_distance();
          find_min_junction();
        }
    }
  }
  return(min_junction);
}
```

Εικόνα 16: Ψευδοκώδικας εύρεσης κόμβου προορισμού.

4. Προτεινόμενοι Αλγόριθμοι

4.1 Αλγόριθμος 1: Παράδοση πακέτων σε μια περιοχή

Ο πρώτος αλγόριθμος θεωρεί ολόκληρο το γράφημα με τα junctions και τους προορισμούς των πακέτων σαν μια περιοχή. Η διαδικασία που ακολουθείται για αυτόν τον αλγόριθμο είναι αρχικά ταξινόμηση των προορισμών των πακέτων βάσει του βάρους τους, δηλαδή από το πιο βαρύ στο λιγότερο βαρύ πακέτο. Στη συνέχεια, γνωρίζοντας με ποια σειρά θα εξυπηρετηθούν οι προορισμοί, για κάθε προορισμό πρέπει να βρεθεί σε ποιον κόμβο πρέπει να μεταφερθεί το φορτηγό. Αρχικά γίνεται έλεγχος αν έχει εκτελεστεί ο αλγόριθμος Dijkstra και το διάνυσμα prev[] δεν είναι κενό. Αν δεν έχει εκτελεστεί ο Dijkstra και ο τρέχον κόμβος δεν έχει εξερχόμενους δρόμους τότε το φορτηγό βρίσκεται σε αδιέξοδο και θα πρέπει να πάει προς τα πίσω μέχρι να βρει κόμβο που να έχει περισσότερους από έναν εξερχόμενους δρόμους. Στην περίπτωση που ο αλγόριθμος Dijkstra έχει εκτελεσθεί και υπάρχουν μονοπάτια συντομότερης διαδρομής, όμως πάλι το φορτηγό βρίσκεται σε κόμβο που δεν έχει εξερχόμενους δρόμους τότε το φορτηγό θα πάει πίσω σε κάποιο κόμβο που να έχει παραπάνω από έναν εξερχόμενους δρόμους σύμφωνα με το διάνυσμα prev[]. Όταν διασφαλιστεί ότι το φορτηγό δεν βρίσκεται σε αδιέξοδο θα μεταβεί σε αυτόν τον κόμβο και θα υπολογιστεί η απόσταση που έχει διανύσει μέχρι αυτό το σημείο. Στην απόσταση αυτή συμπεριλαμβάνονται και οι διαδρομές που κάνει το φορτηγό όταν βρίσκεται σε αδιέξοδο, μέχρι να βρει τον κατάλληλο κόμβο. Από αυτόν τον κόμβο θα ξεκινήσει την πτήση του το drone για την παράδοση του πακέτου στον προορισμό. Κατά την παράδοση του πακέτου γίνεται υπολογισμός της ενέργειας που έχει καταναλώσει το drone μέχρι εκείνη τη στιγμή καθώς και πόσες ώρες πτήσεις έχει συμπληρώσει. Αφού γίνει η παράδοση του πακέτου το drone θα επιστρέψει στο φορτηγό. Πριν το φορτηγό αναχωρήσει για τον επόμενο προορισμό γίνεται έλεγχος αν για τον συγκεκριμένο κόμβο υπάρχουν και άλλα πακέτα προς παράδοση μελλοντικά, ώστε να μην χρειαστεί το φορτηγό να επιστρέψει στην ίδιο κόμβο. Αν υπάρχουν επιπλέον πακέτα προς παράδοση το drone θα κάνει διαδοχικά όσες παραδόσεις αντιστοιχούν σε αυτόν τον κόμβο υπολογίζοντας κάθε φόρα, την κατανάλωση ενέργειας και τις ώρες πτήσεις όπως αναφέρθηκε και παραπάνω. Όταν οι παραδόσεις από αυτόν τον κόμβο έχουν ολοκληρωθεί το φορτηγό μεταβαίνει στον επόμενο κόμβο - προορισμό και επαναλαμβάνει την παραπάνω διαδικασία. Η διαδικασία θα ολοκληρωθεί όταν δεν υπάρχουν πλέον πακέτα για παράδοση. Στο τέλος

του αλγορίθμου υπολογίζεται και το συνολικό throughput των πακέτων, δηλαδή πόσα πακέτα παραδόθηκαν σε μια ώρα.

Ο τύποι για τον υπολογισμό του throughput, της κατανάλωσης ενέργειας και της διάρκειας πτήσης είναι οι εξής:

$$throughput = \frac{number\ of\ destinations}{total\ flight\ time\ (in\ hours)}$$

$$Energy(Watt) = \frac{kg * m^2}{s^3} = \frac{drone\ speed^3 * (packet\ weight + 2 * drone\ weight)}{distance}$$

$$flight\ time = \frac{2 * distance}{drone\ speed}$$

Η ταχύτητα του drone κατά μέσο όρο θεωρήθηκε ότι είναι 15 km/hour.

4.2 Αλγόριθμος 2: Διαίρεση γραφήματος σε περιοχές και παράδοση πακέτων ανά περιοχή

Για τον δεύτερο αλγόριθμο έγινε διαίρεση του γραφήματος σε περιοχές. Από το αρχείο εισόδου του προγράμματος δίνεται ένας αριθμός n για τον διαχωρισμό του γραφήματος σε n x n περιοχές, π.χ αν n=1, θα δημιουργηθεί 1 περιοχή, αν n=2 θα δημιουργηθούν 4 περιοχές. Στη συνέχεια, για κάθε περιοχή που δημιουργείται γίνεται έλεγχος ποια πακέτα ανήκουν στην περιοχή και πάλι όπως στον άλλο αλγόριθμο ταξινομούνται βάσει του βάρους τους. Τέλος, οι περιοχές ταξινομούνται βάσει του συνολικού τους βάρους και έτσι η παράδοση των πακέτων θα ξεκινήσει από την βαρύτερη προς την ελαφρύτερη περιοχή. Για κάθε περιοχή η διαδικασία που ακολουθείται για την παράδοση των πακέτων είναι όμοια με αυτή που περιγράφηκε παραπάνω.

Ο ψευδοκώδικας υλοποίησης των δύο αλγορίθμων είναι κοινός καθώς οι δύο αλγόριθμοι ακολουθούν την ίδια διαδικασία για την παράδοση των πακέτων και αυτό που αλλάζει είναι ο αριθμός των περιοχών. Έτσι όπως βλέπουμε στον ψευδοκώδικα παρακάτω διατρέχονται οι περιοχές και για

κάθε περιοχή ακολουθείται η διαδικασία που περιγράφηκε παραπάνω. Όταν έχουμε μια περιοχή, δηλαδή είμαστε στον πρώτο αλγόριθμο η επανάληψη θα εκτελεστεί μόνο μια φορά.

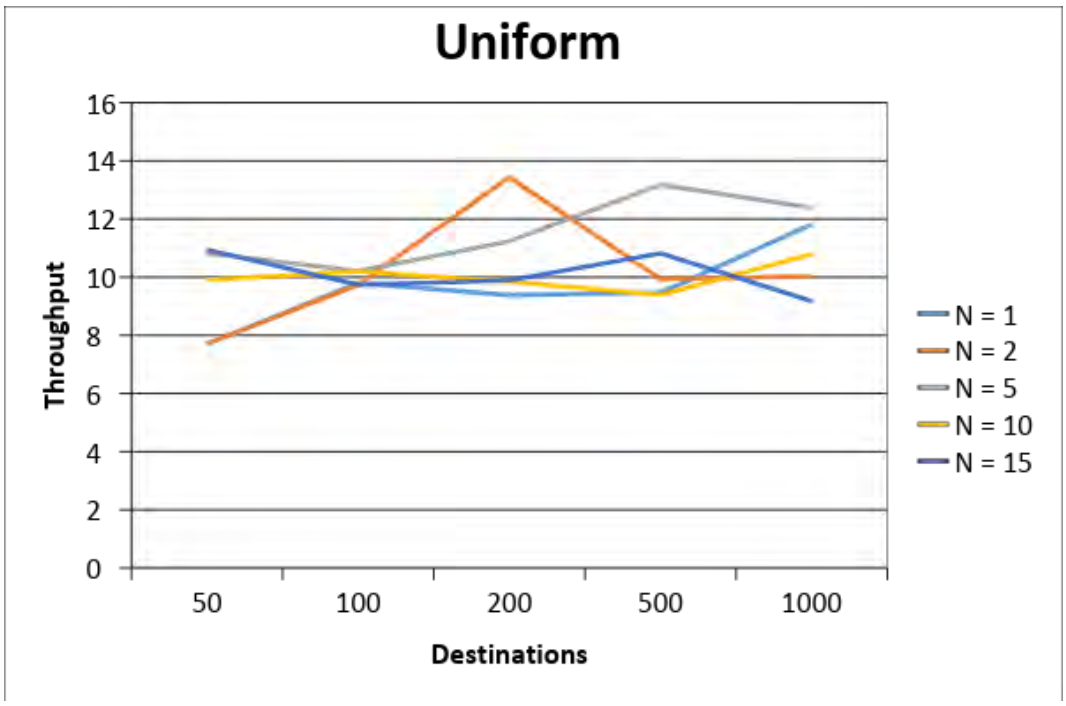
```
function send_one_packet_at_a_time() {

    for each area {
        for each packet_destination{
            if(prev is empty && curr_junction_outcomingRoads == 0){

                while(curr_junction_outcomingRoads <= 1){
                    curr_junction = go_back();

                }

            }else if(prev is not empty && curr_junction_outcomingRoads == 0){

                while(curr_junction_outcomingRoads <= 1){
                    curr_junction = prev[curr_junction];

                }
            }

            Dijkstra_algorithm();
            find_closest_junction_from_destination();
            calculate_distance_covered_by_drone_carrier();
            deliver_packet_to destination();
            calculate_energy_consumption();
            calculate_flight_time();

            if (exists more packets for delivery in this junction){
                deliver_packet_to_destination();
                calculate_energy_consumption();
                calculate_flight_time();

            }

        }
    }

    print total energy consumption;
    print total flight time;
    print packets throughput;

}
```

Εικόνα 17: Ψευδοκώδικας αλγορίθμων 1 & 2.

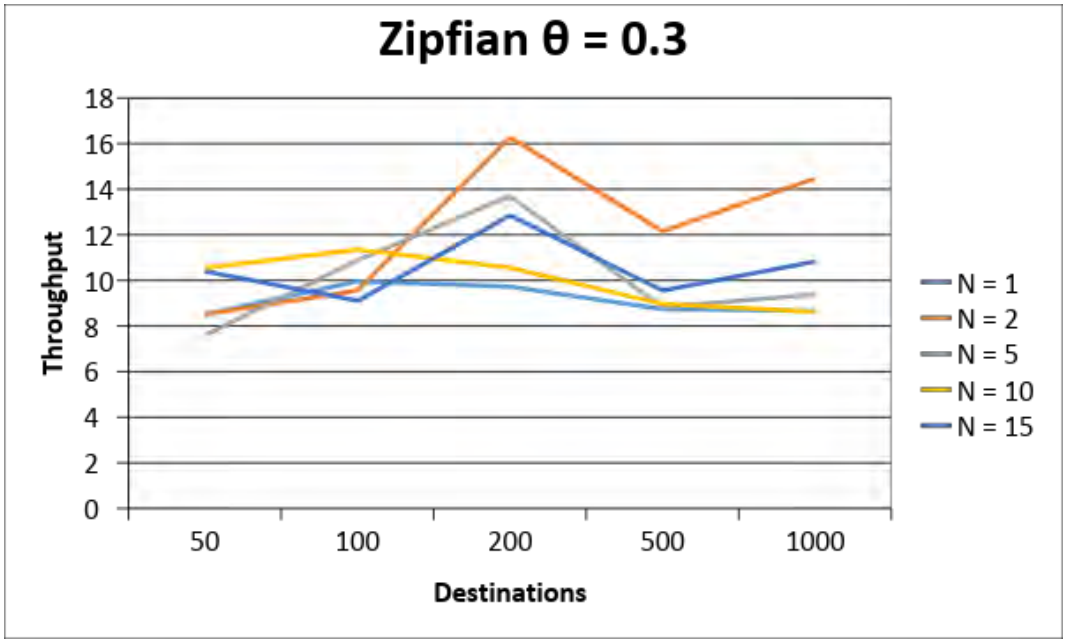
4.3 Επίδοση αλγορίθμων

Αφού ολοκληρώθηκε ο κώδικας του προγράμματος έγιναν πειράματα θέτοντας διαφορετικές τιμές για τον αριθμό των προορισμών των πακέτων, τον αριθμό των περιοχών καθώς και διαφορετικοί τρόποι για την δημιουργία των προορισμών και των πακέτων.
Τα πειράματα εκτελέστηκαν σε ένα μεγάλο γράφημα με 346 κόμβους και αφορούν την ενέργεια που κατανάλωσε το drone για την διανομή όλων των πακέτων, τις ώρες πτήσεις που συμπλήρωσε και το throughput των πακέτων. Τα αποτελέσματα κάθε πειράματος καταχωρήθηκαν σε έναν πίνακα στο excel ώστε στη συνέχεια να γίνει η γραφική αναπαράσταση τους.

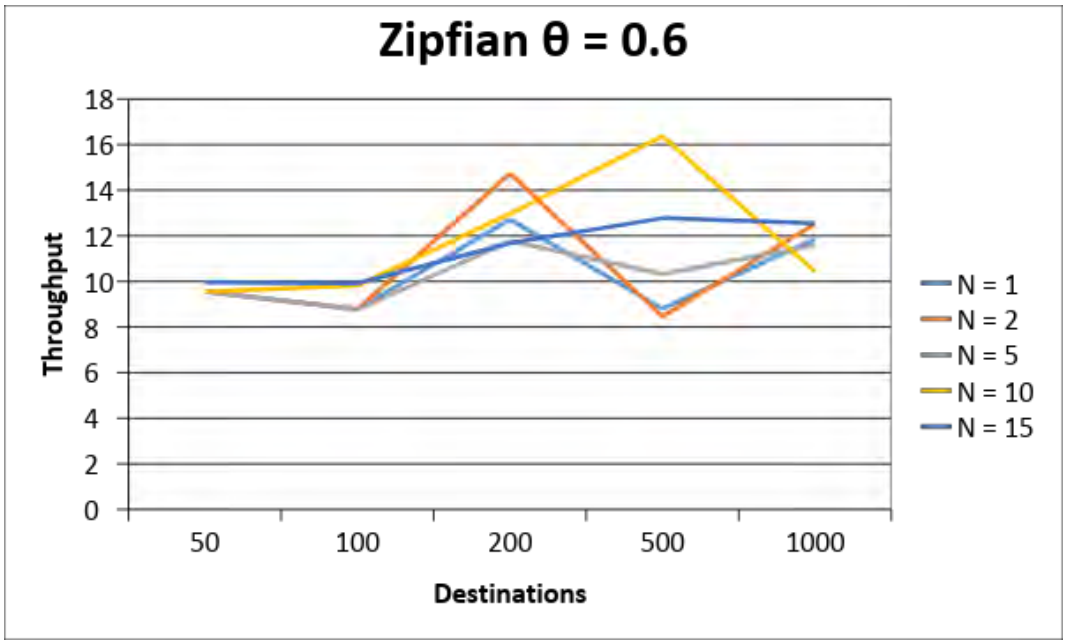
Στα παρακάτω σχήματα βλέπουμε τις γραφικές αναπαραστάσεις με τα αποτελέσματα για διαδοχικές εκτελέσεις του κώδικα. Ο αριθμός των προορισμών που δημιουργήθηκαν είναι: 50, 100, 200, 500, 1000 και το γράφημα χωρίστηκε σε 1 (N=1), 4 (N=2), 25 (N=5), 100 (N=10), 225 (N=15) περιοχές.



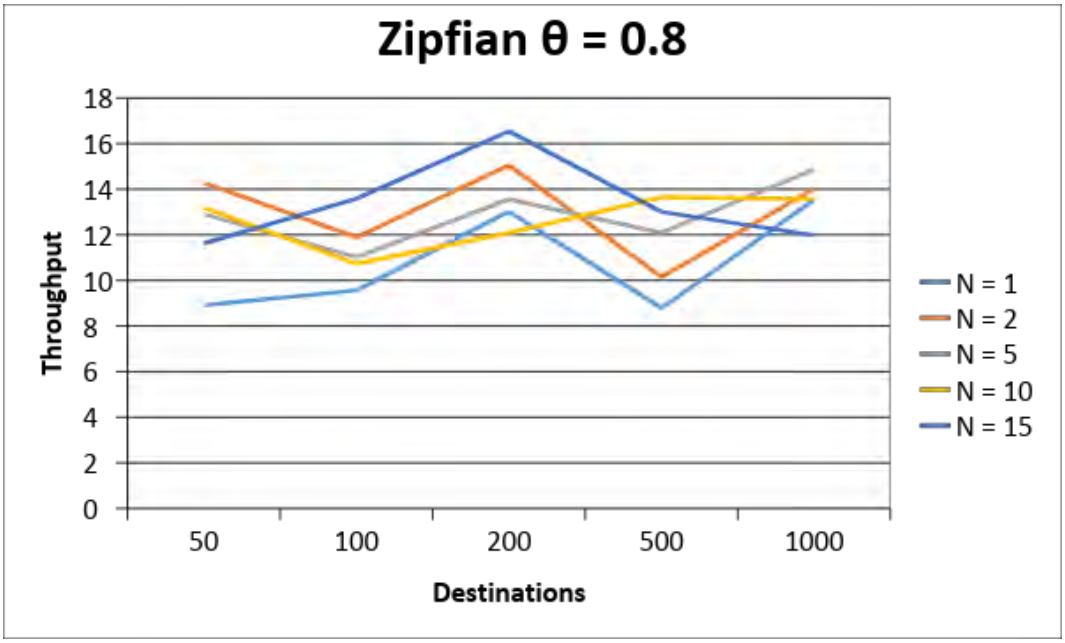
Σχήμα 1: Γραφική αναπαράσταση του throughput των πακέτων για uniform κατανομή.



Σχήμα 2: Γραφική αναπαράσταση του throughput των πακέτων για zipfian κατανομή με θ=0.3.



Σχήμα 3: Γραφική αναπαράσταση του throughput των πακέτων για zipfian κατανομή με θ=0.6.



Σχήμα 4: Γραφική αναπαράσταση του throughput των πακέτων για zipfian κατανομή με θ=0.8.

Στα σχήματα 1, 2, 3 και 4 βλέπουμε την επίδοση των αλγορίθμων σχετικά με το throughput των πακέτων. Καλύτερος αλγόριθμος θεωρείται εκείνος που έχει μεγαλύτερο throughput, δηλαδή εκείνος που παρέδωσε περισσότερα πακέτα σε μία ώρα.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 15
zipfian θ = 0.3	N = 10
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 2

Πίνακας 1: Επίδοση αλγορίθμων σχετικά με το throughput για 50 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 10
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 15

Πίνακας 2: Επίδοση αλγορίθμων σχετικά με το throughput για 100 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 2
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 2
zipfian θ = 0.8	N = 15

Πίνακας 3: Επίδοση αλγορίθμων σχετικά με το throughput για 200 προορισμούς.

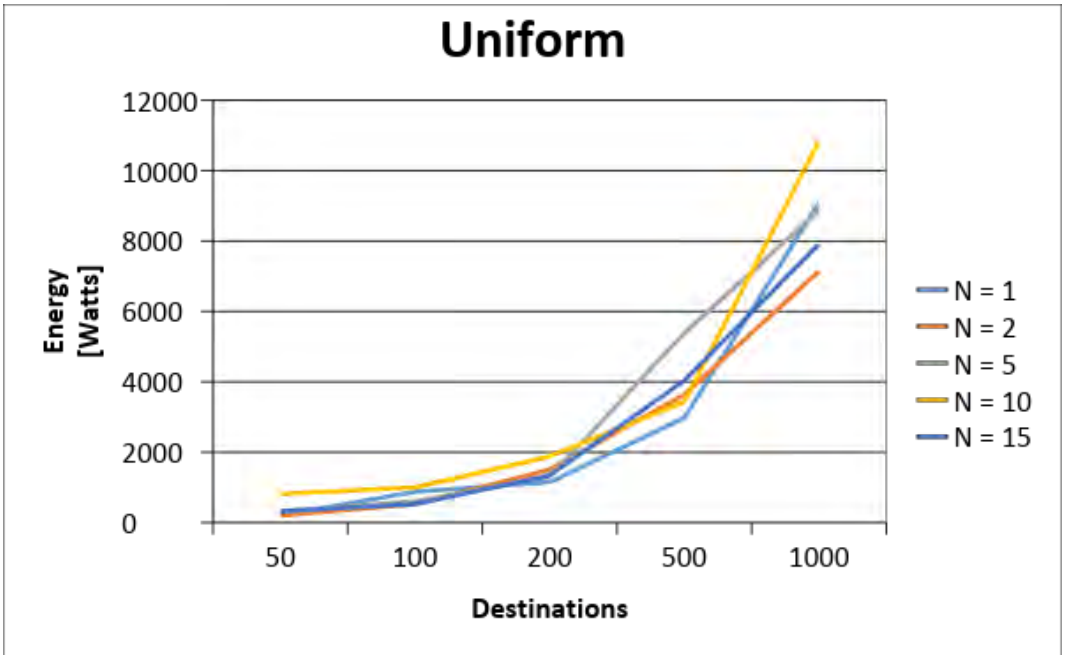
κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 10
zipfian θ = 0.8	N = 10

Πίνακας 4: Επίδοση αλγορίθμων σχετικά με το throughput για 500 προορισμούς.

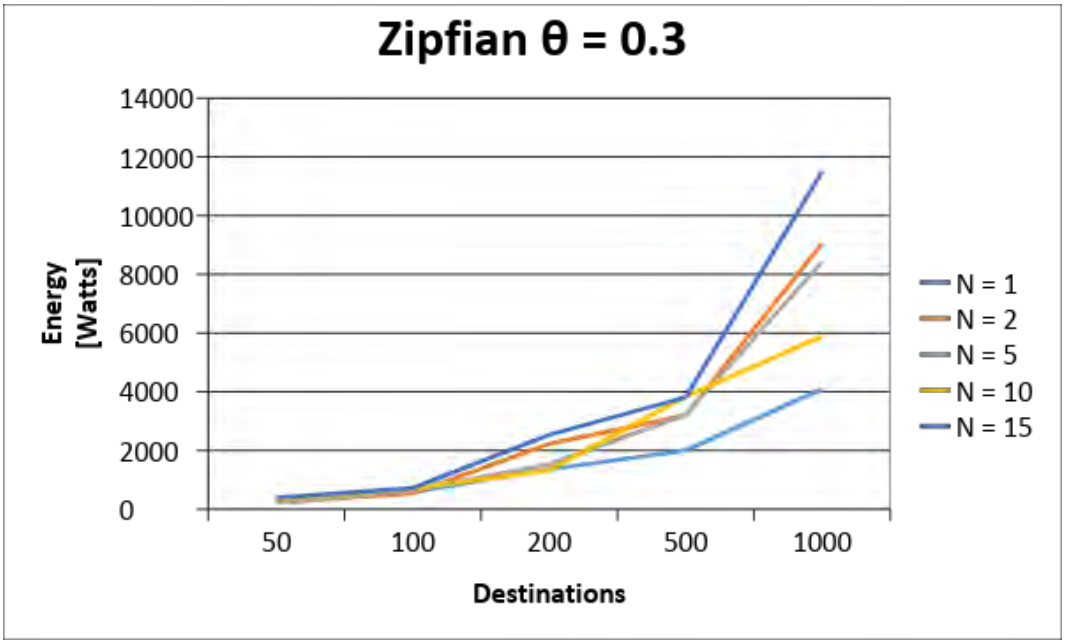
κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 5

Πίνακας 5: Επίδοση αλγορίθμων σχετικά με το throughput για 1000 προορισμούς.

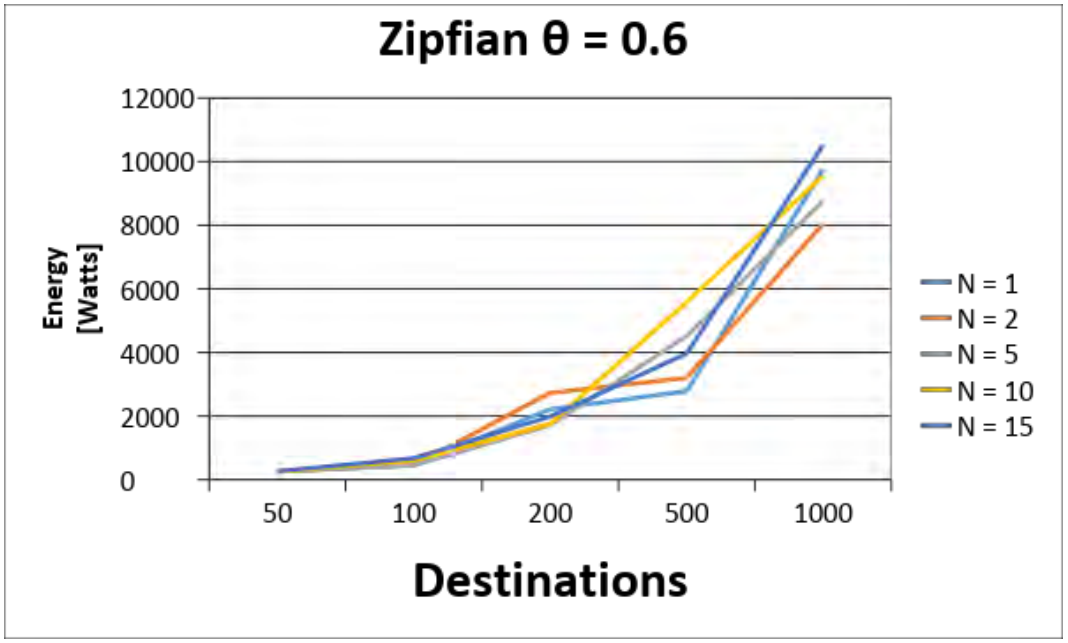
Όπως παρατηρούμε από τους παραπάνω πίνακες διαπιστώνουμε ότι ο αλγόριθμος 1, δηλαδή για N = 1 δεν είναι καλύτερος όσον αφορά το throughput των πακέτων. Πάντα καλύτερος είναι ο αλγόριθμος 2 για διαφορετικά N. Όμως δεν μπορούμε να πούμε ξεκάθαρα ποια είναι η καλύτερη τιμή του N για να είναι πιο αποδοτικός ο αλγόριθμος 2.



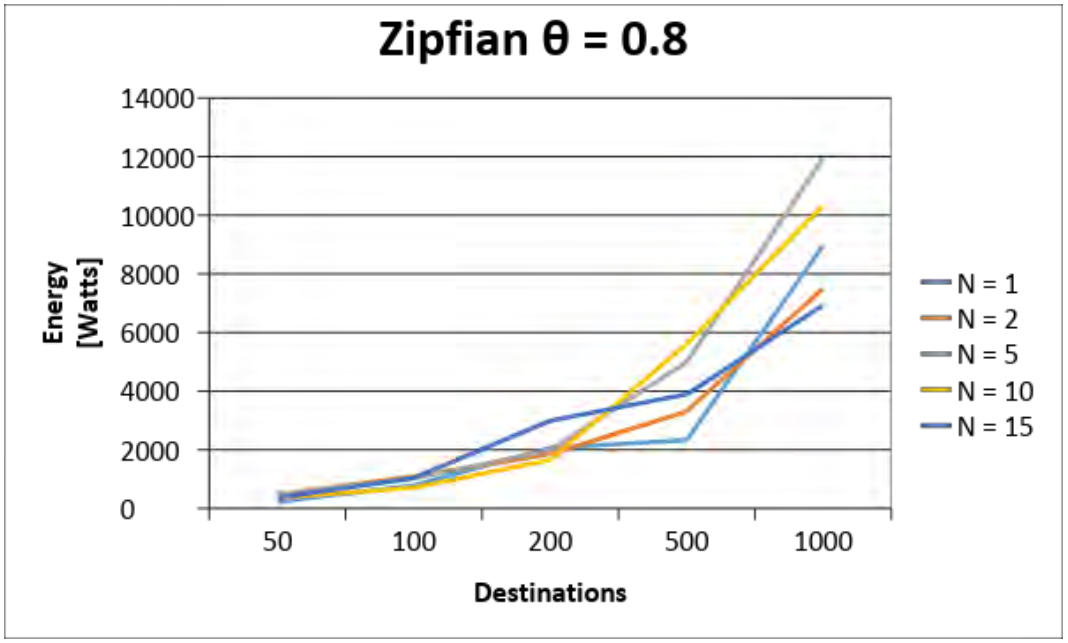
Σχήμα 5: Γραφική αναπαράσταση της ενέργειας που καταναλώθηκε για uniform κατανομή.



Σχήμα 6: Γραφική αναπαράσταση της ενέργειας που καταναλώθηκε για zipfian κατανομή με θ=0.3.



Σχήμα 7: Γραφική αναπαράσταση της ενέργειας που καταναλώθηκε για zipfian κατανομή με θ=0.6.



Σχήμα 8: Γραφική αναπαράσταση της ενέργειας που καταναλώθηκε για zipfian κατανομή με θ=0.8.

Στα σχήματα 5, 6, 7, και 8 βλέπουμε την επίδοση των αλγορίθμων σχετικά με την κατανάλωση ενέργειας. Καλύτερος αλγόριθμος θεωρείται εκείνος που κατανάλωσε το μικρότερο ποσό ενέργειας. Από τη μορφή των γραφικών παραστάσεων φαίνεται σαν η ενέργεια να αυξάνεται εκθετικά. Όμως για να μπορούμε να πούμε ότι είναι εκθετική πρέπει για σταθερή

αύξηση της ανεξάρτητης μεταβλητής που είναι ο αριθμός destinations να προκαλείται σταθερή ποσοστιαία αύξηση και της εξαρτημένης μεταβλητής που είναι το ποσό της ενέργειας. Εφόσον, ο αριθμός των destinations δεν αυξάνεται με σταθερό ρυθμό δεν μπορούμε να εξάγουμε συμπέρασμα για τον τρόπο αύξησης των καμπυλών.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 1 , N = 2
zipfian θ = 0.3	N = 5
zipfian θ = 0.6	N = 1, N=2, N=5, N=10
zipfian θ = 0.8	N = 1

Πίνακας 6: Επίδοση αλγορίθμων σχετικά με την κατανάλωση ενέργειας για 50 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 2, N =15
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 1, N=2, N=5
zipfian θ = 0.8	N = 10

Πίνακας 7: Επίδοση αλγορίθμων σχετικά με την κατανάλωση ενέργειας για 100 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 1
zipfian θ = 0.3	N = 10
zipfian θ = 0.6	N = 5
zipfian θ = 0.8	N = 10

Πίνακας 8: Επίδοση αλγορίθμων σχετικά με την κατανάλωση ενέργειας για 200 προορισμούς.

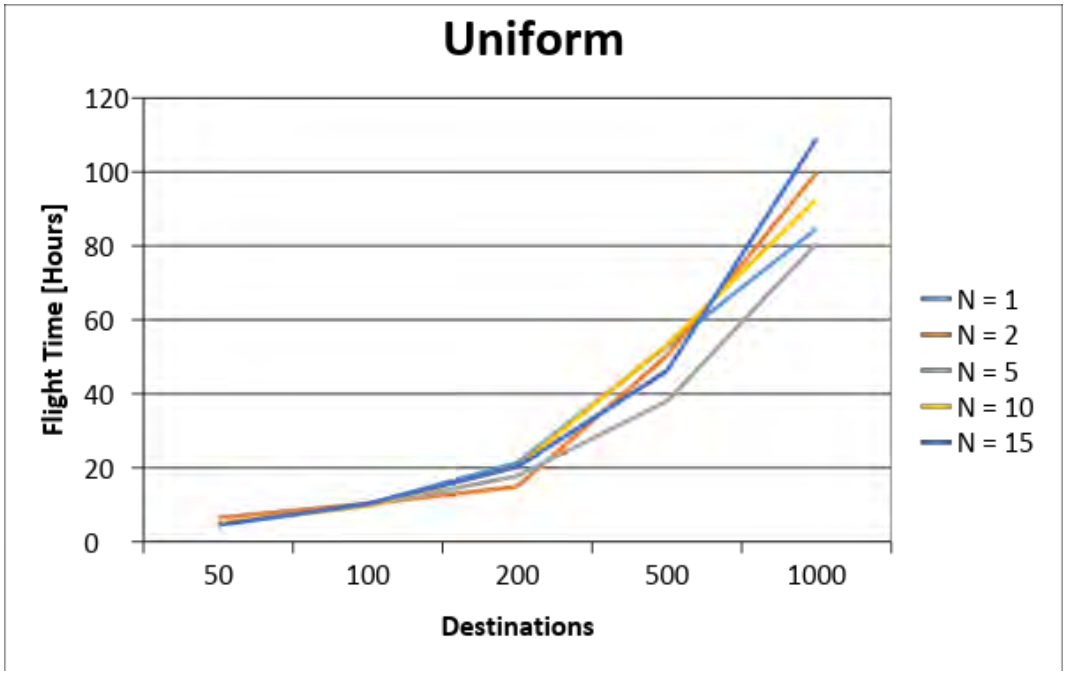
κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 1
zipfian θ = 0.3	N = 1
zipfian θ = 0.6	N = 1
zipfian θ = 0.8	N = 1

Πίνακας 9: Επίδοση αλγορίθμων σχετικά με την κατανάλωση ενέργειας για 500 προορισμούς.

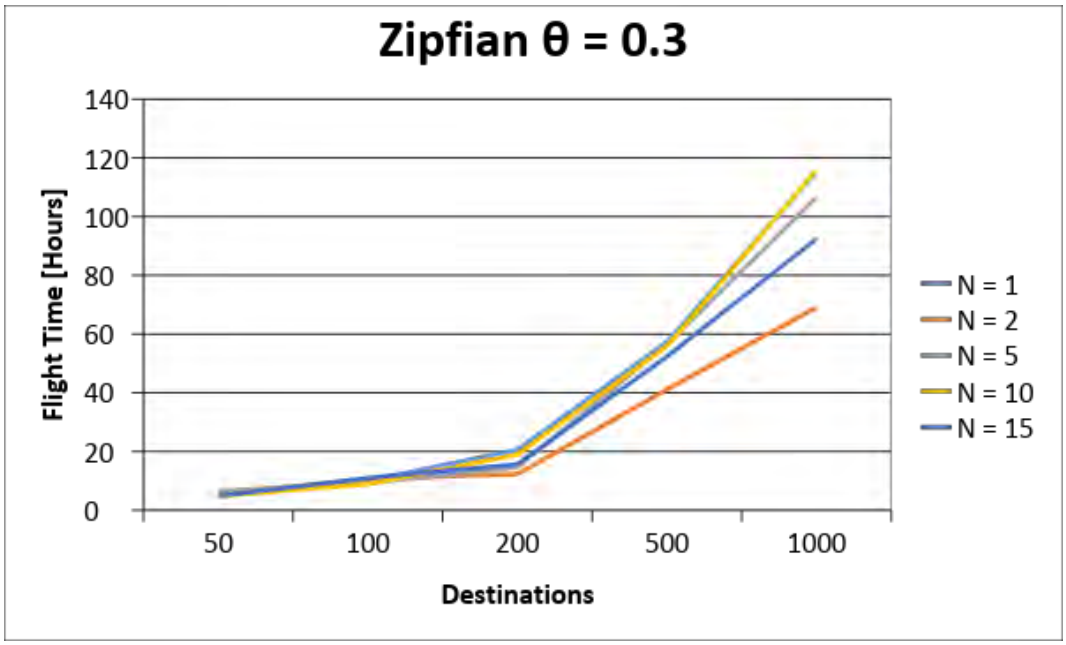
κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 2
zipfian θ = 0.3	N = 1
zipfian θ = 0.6	N = 2
zipfian θ = 0.8	N = 15

Πίνακας 10: Επίδοση αλγορίθμων σχετικά με την κατανάλωση ενέργειας για 1000 προορισμούς.

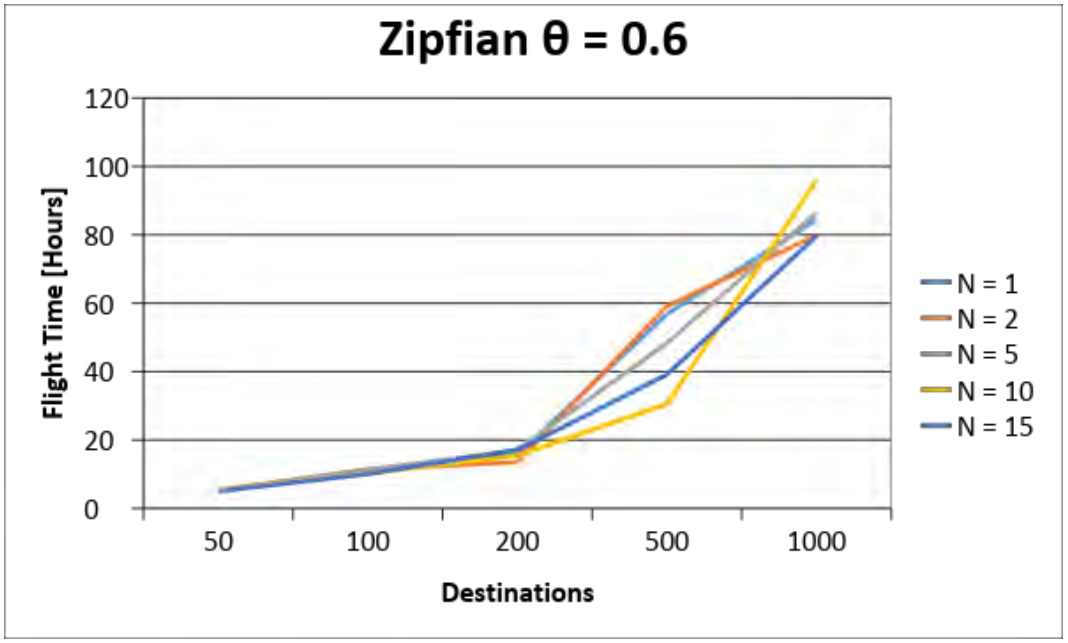
Όπως παρατηρούμε από τους παραπάνω πίνακες σχετικά με την κατανάλωση ενέργειας για 500 προορισμούς καλύτερος αλγόριθμος σε όλες τις κατανομές είναι ο αλγόριθμος 1, δηλαδή για N=1. Στις άλλες περιπτώσεις είναι διαφορετικός αλγόριθμος καλύτερος για διαφορετικές κατανομές, όμως θα μπορούμε να πούμε ότι ο αλγόριθμος 2, δηλαδή για N>1 υπερτερεί.



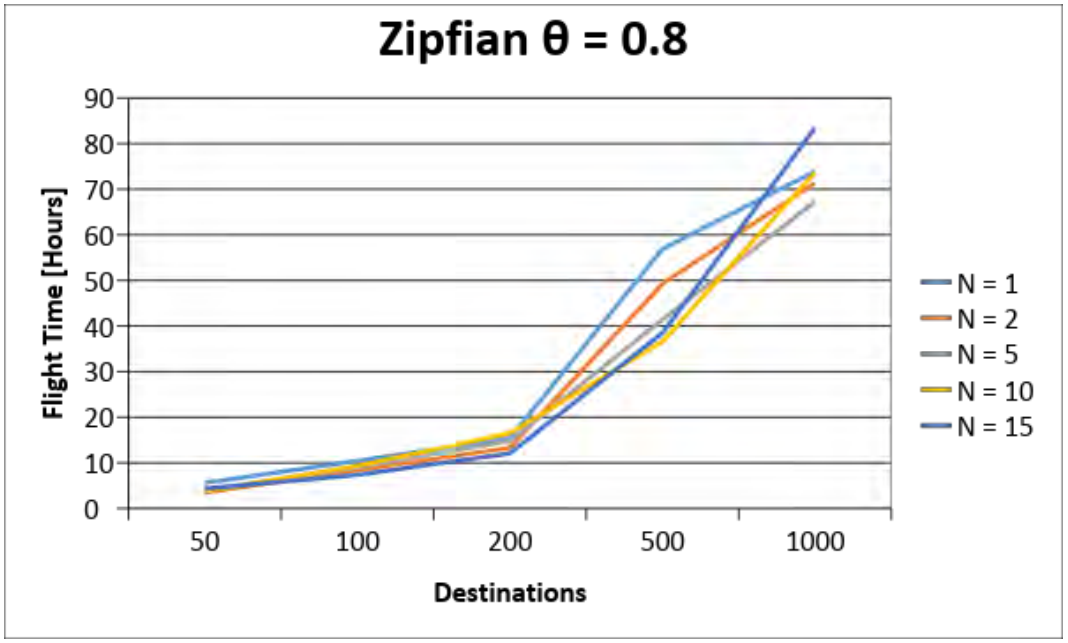
Σχήμα 9: Γραφική αναπαράσταση του χρόνου πτήσης για uniform κατανομή.



Σχήμα 10: Γραφική αναπαράσταση του χρόνου πτήσης για zipfian κατανομή με θ=0.3.



Σχήμα 11: Γραφική αναπαράσταση του χρόνου πτήσης για zipfian κατανομή με θ=0.6.



Σχήμα 12: Γραφική αναπαράσταση του χρόνου πτήσης για zipfian κατανομή με θ=0.8.

Στα σχήματα 9, 10, 11, 12 βλέπουμε την επίδοση των αλγορίθμων σχετικά με τις ώρες πτήσεις. Καλύτερος αλγόριθμος είναι αυτός που έχει συμπληρώσει λιγότερες ώρες πτήσεις. Όμοια με τις γραφικές παραστάσεις της ενέργειας και εδώ δεν μπορούμε να εξάγουμε ένα σαφές συμπέρασμα για τον τρόπο αύξησης των καμπυλών.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 15
zipfian θ = 0.3	N = 10
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 2

Πίνακας 11: Επίδοση αλγορίθμων σχετικά με το χρόνο πτήσης για 50 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 10
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 15

Πίνακας 12: Επίδοση αλγορίθμων σχετικά με το χρόνο πτήσης για 100 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 2
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 2
zipfian θ = 0.8	N = 15

Πίνακας 13: Επίδοση αλγορίθμων σχετικά με το χρόνο πτήσης για 200 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 10
zipfian θ = 0.8	N = 10

Πίνακας 14: Επίδοση αλγορίθμων σχετικά με το χρόνο πτήσης για 500 προορισμούς.

κατανομή προορισμών	**καλύτερη επίδοση**
uniform	N = 5
zipfian θ = 0.3	N = 2
zipfian θ = 0.6	N = 15
zipfian θ = 0.8	N = 5

Πίνακας 15: Επίδοση αλγορίθμων σχετικά με το χρόνο πτήσης για 1000 προορισμούς.

Όπως παρατηρούμε από τους παραπάνω πίνακες διαπιστώνουμε ότι ο αλγόριθμος 1, δηλαδή για N = 1 δεν είναι καλύτερος όσον αφορά το flight time. Πάντα καλύτερος είναι ο αλγόριθμος 2 για διαφορετικά N. Όμως και σε αυτή την περίπτωση δεν μπορούμε να πούμε ξεκάθαρα ποια είναι η καλύτερη τιμή του N για να είναι πιο αποδοτικός ο αλγόριθμος 2.

5. Σύνοψη

5.1 Συμπεράσματα

Στόχος αυτής της εργασίας ήταν η ανάπτυξη αλγορίθμων για την διανομή πακέτων με χρήση UAV σε συνεργασία με ένα φορτηγό. Οι αλγόριθμοι που αναπτύχθηκαν ήταν δύο και είχαν να κάνουν με το διαχωρισμό του γραφήματος σε υποπεριοχές. Στον πρώτο αλγόριθμο, ολόκληρο το γράφημα με το οδικό δίκτυο και τους προορισμούς θεωρήθηκε σαν μια περιοχή και η διανομή των πακέτων έγινε διαδοχικά από το βαρύτερο πακέτο προς το λιγότερο βαρύ. Στον δεύτερο αλγόριθμο, έγινε διαχωρισμός του γραφήματος σε μικρότερες υποπεριοχές, όπου οι υποπεριοχές ταξινομήθηκαν βάσει του συνολικού βάρους των πακέτων και η διανομή έγινε από την πιο βαριά περιοχή στην λιγότερο βαριά. Οι αλγόριθμοι μετράνε το συνολικό throughput των πακέτων, την κατανάλωση ενέργειας του UAV και τον χρόνο πτήσης του UAV. Στη συνέχεια, μελετώντας την επίδοση των αλγορίθμων παρατηρούμε ότι για το χρόνο πτήσης και το throughput των πακέτων καλύτερος είναι για όλες τις περιπτώσεις ο δεύτερος αλγόριθμος, όμως για διαφορετικό αριθμό περιοχών ανά περίπτωση. Ενώ σχετικά με την κατανάλωση ενέργειας ανάλογα τις περιπτώσεις είναι καλύτερος είτε ο πρώτος είτε ο δεύτερος αλγόριθμος. Το συμπέρασμα που προκύπτει είναι ότι δεν υπάρχει ξεκάθαρη απάντηση για το ποιος αλγόριθμος είναι καλύτερος γενικά σε όλες τις περιπτώσεις. Ακόμη και όταν προκύπτει ο δεύτερος αλγόριθμος καλύτερος σε όλες τις περιπτώσεις πρέπει να προσδιοριστεί και σε πόσες υποπεριοχές έχει χωριστεί το γράφημα.

5.2 Μελλοντική Έρευνα

Σαν μελλοντική εργασία θα μπορούσε να γίνει έρευνα σε ένα πολύ μεγαλύτερο γράφημα και με περισσότερα UAVs που θα συνεργάζονται με ένα φορτηγό και θα κάνουν διανομές πακέτων σε κοντινούς προορισμούς παράλληλα. Επιπλέον, θα μπορούσε να γίνονται αναθέσεις διανομών από ένα UAV σε άλλο, όταν το πρώτο δεν θα έχει ενέργεια για να εκτελέσει τις διανομές του.

Βιβλιογραφία

[1] UAVs: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle#Power_supply_and_platform

[2] Quadcopters: https://en.wikipedia.org/wiki/Quadcopter

[3] Amazon: https://www.bizjournals.com/seattle/news/2018/03/27/driverless-delivery-amazon-patent-trucks-drones.html

[4] Drones and environment: http://theconversation.com/delivering-packages-with-drones-might-be-good-for-the-environment-90997

[5] Σελίδα SUMO simulator: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[6] Wiki SUMO: http://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki

[7] OpenStreetMap: https://www.openstreetmap.org/#map=15/39.3571/22.9475

[8] Πληροφορίες για Dijkstra: https://en.wikipedia.org/wiki/Dijkstra's_algorithm

[9] Giorgio Gallo, Stefano Pallotino (1986) "Shortest Path Methods: A Unifying Approach"